

A Novel System-Call Dataset for AI-Driven Host-Based Intrusion Detection in Containerized Cloud Environments

AMMAR MOUSTAFA

`ammam.c.moustafa@latakia-univ.edu.sy`

Wadi International University <https://orcid.org/0009-0006-7361-3272>

Short Report

Keywords: Host-Based Intrusion Detection, System Call Dataset, Anomaly Detection, Container Security, Cloud Computing

Posted Date: October 19th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11351546/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: The authors declare no competing interests.

A Novel System-Call Dataset for AI-Driven Host-Based Intrusion Detection in Containerized Cloud Environments

1st Ammar Moustapha

*Dept. Informatics Engineering, Faculty of
Engineering
Wadi International University
Homs, Syria
ammarmoustapha@gmail.com*

2nd Ali Younes

*Dept. Information Engineering, Computer Science
and Mathematics
University of L'Aquila
L'Aquila, Italy
ali.younes@student.univaq.it*

3rd Balsam Eid

*Dept. Computer Engineering
Latakia University
Latakia, Syria
Balsameid72@gmail.com*

Research Draft

Abstract— Host-based intrusion detection in environments needs datasets that show how system calls behave and the situation in which they happen. Current system-call datasets were made for Linux systems and don't cover many container-aware tasks or new types of attacks. This paper introduces CHASE (Container Host Attack and System-call Execution Dataset) which has 4,980 traces: 4,482 normal and 498 harmful. Each trace includes 105 features like time, structure, actions, resources and behavior such as system calls, process and thread details, timestamps, resource use I/O and network activity and container context. The harmful traces include 22 instances from nine attack groups, based on CVE and scenarios. There is also a Synthetic Unseen group for testing new situations. We check the dataset using analysis of data leakage, group-aware testing time shift tests, leave-one-family-out evaluation, testing across environments and checking the importance of each feature. Among the models tested Random Forest gets a PR-AUC of 0.9558 and a ROC-AUC of 0.9845. Testing across environments gives a PR-AUC of 0.9900 and leave-one-family-out evaluation gets a PR-AUC of 1.0000, for some attack families that were not seen before. The results show that CHASE helps with detection testing and checking how well models can handle attack families they didn't see during training.

Keywords —Host-Based Intrusion Detection, System Call Dataset, Anomaly Detection, Container Security, Cloud Computing

I. INTRODUCTION

Containerized computing is now widely used for launching applications in cloud and distributed settings. Because containers are light and portable they make deployment easier. Yet containers share the host operating-system kernel. As a result they do not give the level of isolation that virtual machines do. This lack of isolation opens the door to attacks that target kernel bugs, excess privileges, insecure images or mistakes in configuration. These risks have made people more interested in monitoring the host as a defense for containerized systems [1] [2]. Host-based intrusion detection systems, known as HIDS can spot activity by watching the operating-system's behavior. System calls are especially helpful for this because they reveal how applications talk to the kernel. Recent studies have looked at sequences of system calls frequency-based representations and machine-learning techniques for HIDS in both containerized environments [3]–[5]. Other research has shown that adding execution details, such as which thread made the call can improve detection. This is because system calls from threads can get mixed together [13]. With these advances it is still hard to compare results from different HIDS studies. The difficulty comes from the fact that outcomes depend a lot on the datasets, workloads and evaluation methods that are used. The problem is especially noticeable in settings, where many applications, services, containers and host processes run at the same time. A record of system calls that lacks execution context can lose the information about which process or workload generated the event and under what conditions it happened. The issue is not, about the detection algorithm; it also questions whether the benchmark supplies enough data to allow a meaningful evaluation of behavioral detection.

CHASE closes this gap by bringing system-call behavior with details, about the execution environment like timing, process and thread data, resource use I/O actions and network activity. It also adds container metadata. To test how well detection models work in real-world situations it includes background workloads, hard negative examples and carefully designed tests that expose behaviors not seen during training. This helps measure how well models can adapt to unseen situations.

II. RELATED WORK

Early HIDS research set system-call traces as a source for spotting odd program behavior. The DARPA evaluations gave a framework for systematic and repeatable IDS testing[7], but later work showed that old datasets were not enough and that new data that better matches real life was needed[8]. ADFA-IDS then offered Linux and Windows collections for system-call-based HIDS study. After that researchers looked at ways to show system-call behavior[9]. Creech and Hu looked at ways to describe system-call patterns[10], while Pendleton and Xu found that flat syscall sets were limited and built a generator that adds structure and context[11]. Using learning pushed system-call-based HIDS further. Ring and colleagues developed ALAD, a learning anomaly detector, for system-call sequences and tested it on ADFA-LD and PLAID[12]. Newer work also looks at container settings. Joraviya and team introduced DL-HIDS, which turns system-call data into pictures and tests CNN models on LID-DS 2019[1]. In work their Ab-HIDS method uses N-gram system-call frequencies along with ensemble learning to find unusual behavior in containerized environments [2]. Vyšniūnas and others looked at grouping system-call sequences based on risk. They found that simple classifiers like SVM and decision trees can do a good job of detecting malware [3]. Chen and others

created a HIDS that brings together information from the host and the network and they stressed how important it is to use different sources of evidence [4]. At the level of datasets and testing Satilmis and others looked at HIDS research and found big differences in the ways detection is done the data that is used and how testing is carried out [5]. Arnoud and others also studied system-call-based IDS research. Pointed out ongoing problems with datasets being able to repeat results and how useful the methods are in real situations [6]. LID-DS 2021 helped make HIDS testing more reliable by giving a way to record data, a benchmark dataset and a tool for testing [14]. Grimmer and others showed that adding thread information can make anomaly detection better in environments where many threads are running [13]. Altogether these studies show that there is progress being made in how system calls are used how learning is done and how datasets are built.. Current methods still don't do a great job of testing HIDS when there are many containers running at the same time. They also don't do a job of including the context of how things are running and checking if the methods work in situations that are different, from what was used during training.

III. RESEARCH GAP AND DESIGN RATIONALE

Despite progress evaluating host-based intrusion detection systems that use system calls still does not have benchmarks that match the complexity of workloads that run in containers at the same time. Earlier research has made improvements in how system calls represented added information about structure and context looked at behavior at the thread level or focused on detecting issues specific to containers [1] [2] [11] [13] [14]. However these features are not often used together. Tested using the same rules that look at changes in workloads, environments and types of attacks. CHASE fills this gap by combining system calls with relationships between processes and threads container context, execution details, resource use and activity related to input and output or network. It also allows for controlled testing using workloads that're harmless and run at the same time hard negatives, analysis of new behaviors and testing methods like leave-one-family-out, temporal-shift, cross-environment and feature-ablation. This makes it possible to get a realistic idea of how well host-based intrusion detection systems can detect threats how well they can adapt and how strong they are, in environments that use containers.

IV. DESCRIPTION OF THE PROPOSED DATASET

To address several of the limitations found in existing datasets, we developed CHASE (Container Host Attack and System-call Execution Dataset) as a controlled behavioral dataset for studying host-based anomaly detection in containerized environments. Then using different types of operational logs that are hard to repeat. CHASE was created in a controlled lab setting. The workload and what was seen to happen were set up to match situations when containers are running. The attack situations were made to focus on security issues that matter for today's container setups. These include known security flaws, weak setups and attacks that target containers and GPU resources. Each situation has an attack type and a step, in the process. If there is a match each situation is connected to a MITRE ATT&CK method. The resulting attack set covers behaviors. Attack set includes container escape. Attack set includes Docker privilege escalation. Attack set includes path traversal. Attack set includes cross-site scripting or XSS. Attack set includes command injection. Attack set includes deserialization. Attack set includes SQL injection. Attack set includes GPU resource hijacking. Attack set includes AI model extraction. The benign portion of CHASE was generated from normal workloads rather than from a minimal background process designed only to provide a negative class. It includes routine interactions among applications, services, containers, and the host system, thereby introducing the variability expected during normal operation. This distinction is important for anomaly detection, since a useful benchmark must expose models to the diversity of legitimate behavior that they are expected to distinguish from malicious activity. it encompasses diverse natural sessions and benign near-miss events, aiming to increase behavioral similarity between certain benign and malicious activities and to reduce detection reliance on superficial indicators. At the level of representation, the generator offers a set of time-based, operation-based and behavior-based features. These include CPU usage and memory usage; how long things take to run; how fast data moves in and out; how much data is moving through the network and memory; how long the queue is; how often the system switches between tasks; how deep the list of processes goes. The details of the sequence of system calls. There is a difference between the features that come straight from data and the features that are created by looking at the behavior and time aspects. Some signs that might be important, for security and could be connected to attacks are explained in an extra layer. Data is stored at levels including event-level telemetry system call sequences, Trace-level labeling was added to help researchers check binary detection and multi-class threat classification accurately. the application stack used in CHASE, including web servers like nginx, Apache, PHP-FPM databases like MySQL, PostgreSQL, Redis and container runtimes such as runc,

Docker, containerd/CRI-O and Build Kit. The environment also has system daemons and AI-related workloads, including Python, torchrun/Deep Speed, Triton Server and NVIDIA GPU monitoring along, with background processes. The dataset has situations linked to CVEs including container escape, web and command injection deserialization Docker issues, AI-model exfiltration and GPU crypto jacking. Figure 1 gives an overview of the dataset.

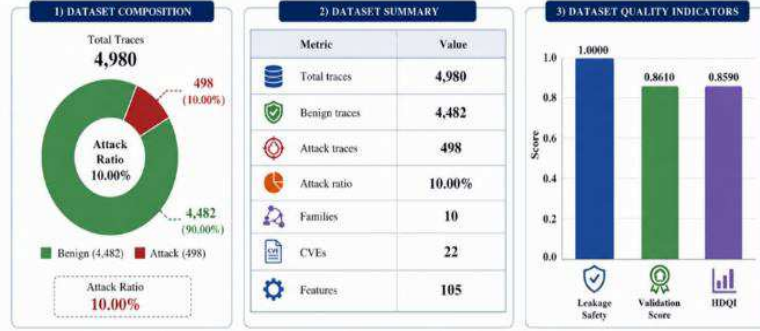


Fig. 1. Summary of the proposed dataset.

V. METHODOLOGY

CHASE was built to keep the connection, between system-call actions and the exact conditions when they happen. Of creating random system-call lists the trace generation follows real data from telemetry. This data shows workload patterns timing details, resource use and the environment where things run. The approach includes making traces that know the context building attacks based on this information organizing features in a way controlling what gets leaked and testing how well it works across different situations.

A. Context-Aware Trace Generation

Trace generation is adjusted based on reference telemetry that captures characteristics GPU utilization, inference latency and event timing. A Causal Coupling Engine connects workload activity to the execution context and available resources showing how they relate to GPU pressure, utilization and inference latency. When possible empirical inter-event time distributions are saved to keep workload- timing behavior. Each trace is shown as an ordered sequence:

$$T = (e_i, t_i, c_i, r_i)_{i=1}^N, \quad (1)$$

Where e_i , t_i , c_i , and r_i denote the system event, timestamp, execution context and resource state respectively and N is the trace length This representation couples system call behavior, with the process workload and resource conditions surrounding each event.

B. Attack Construction and Temporal Embedding

Attack scenarios come from CVE specifications and known system behaviors. For attacks that unfold in steps we use directed acyclic process graphs to represent the sequence. For attacks focused on payloads, like -site scripting, SQL injection and path traversal, we rely on specialized generators. Of replacing normal system activity malicious actions are inserted into an ongoing legitimate workload. This keeps the behavior realistic and harder to detect.

$$T^* = T_{\text{pre}} \oplus A \oplus T_{\text{post}}, \quad (2)$$

The symbol A represents the attack sequence. The attack sequence is inserted at a point that's between twenty percent and sixty percent of the length of the original trace while still following the correct behavioral stages. The timing for each stage of the attack sequence is set using reference distributions of the time between events, when those distributions exist. When those distributions are not available other distributions are used in order to show patterns of speed, slowness, pauses, multiple stages or stealth. The CHASE dataset contains negative traces that are real operations which might look like attacks, at the system call level for example authorized administrative shell work and important file backup tasks. This makes it easier to tell attacks by looking at the context in which the code runs, instead of just looking at single system calls.

C. Feature Organization and Leakage Control

Features are organized into three primary groups: raw telemetry, behavioral characteristics, and temporal characteristics. Raw telemetry includes CPU, memory, GPU, I/O, network activity, context switching, and process depth. Behavioral features capture system-call entropy and diversity, unique-call counts, network-event frequency, file operations, memory allocation, context-switch frequency, and errors. Temporal features include trace length and statistics of inter-event intervals, including mean, standard deviation, minimum, maximum, coefficient of variation, and burstiness. Security-related inferential indicators are maintained separately and excluded from the predictive feature set to avoid label leakage.

D. Data Splitting and Generalization

Group-based evaluation methods were used to check how well CHASE can handle situations that were not part of the training data. The first method, called Leave-One-CVE-Out (LOCO) takes out one CVE vulnerability. The second method, called Leave-One-Family-Out (LOFO) leaves out a group of attacks and then checks how well the model works with that group. The CHASE dataset also has a Synthetic category. This category is used to see if the model can find types of behavior that it has never seen before. This part is meant to test how well the model can detect activity in a controlled setting. It should not be seen as a way to find "zero-day attacks", in life. Also tests are done in environments to make sure the results are the same no matter where the model is used.

E. Assessment of Dataset Utility and Validation of Feature Space

To check if the CHASE dataset offers behavioral information for detecting intrusions we use logistic regression as a basic model. This experiment is not about creating a detection model. It is about seeing if the features we have extracted can tell the difference between harmful activities. Because the dataset has a number of classes we used PR-AUC and ROC-AUC as the main ways to measure how well the model works instead of using accuracy. Also we did a test to see if the difference between known attacks and new fake traces is still there after the features are adjusted. We also wanted to know if this difference comes mainly from things, like how long the traces are or what type of workload they are.

VI. VERIFICATION AND EVALUATION STRATEGY

A. Dataset Verification and Quality Assessment

The reliability of any reference dataset is measured by the rigor of the methodology applied to it. This work adopts a multi-stage verification and evaluation strategy based on the fundamental principle of structure first, utility second: the integrity of the dataset's statistical structure and its independence from leakage are assessed entirely independently of any classifier, followed by a measurement of utility within a strictly controlled experimental protocol that prevents information leakage or adaptation to the test set. The strategy comprises 16 stages executed in a specific sequence, with each stage serving as a quality control checkpoint that must be passed before proceeding to the next. Table I illustrates the assessment methodology.

TABLE I
16-STAGE PIPELINE TO VALIDATE DATASET QUALITY AND MODEL GENERALIZATION.

Stage	Component	Description & Methodology
1-3	Ingestion & Integrity	Trace/label alignment, orphan detection, MinHash-LSH near-duplicate detection & cross-split leakage gate.
4-6	Audit & Realism	Feature Audit (VIF, missingness), Leakage Audit, Distribution (KS/Wasserstein/JSD), and Internal Plausibility checks.
7-8	Group Split & Nested CV	Semantic grouping (Campaign/CVE), Stratified Group 5 × 3 CV across 5 algorithms (RF, IF, SVM, OC-SVM, LSTM).
9-10	OOF Metrics & Threshold	Out-of-Fold PR-AUC/ROC-AUC/MCC, Per-Attack evaluation, Train-derived PR threshold optimization.
11-13	Generalization Triad	Strict Unseen-Behavior Holdout, Leave-One-Family-Out (LOFO), Temporal Shift, Cross-Environment, and Adversarial Validation.
14-16	Ablation & Final Report	Strict Synthetic Unseen Holdout, Leave-One-Family-Out (LOFO), Temporal Shift, Cross-Environment, and Adversarial Validation.

B. Model Evaluation and Decision-Making Protocol

Five models were used for comparison: Random Forest (RF), Support Vector Machine (SVM), Long Short-Term Memory (LSTM), One-Class Support Vector Machine (OCSVM), and Isolation Forest (IF). All models use median imputation and feature scaling fitted on the training data only. RF and SVM hyperparameters are selected through nested 5×3 StratifiedGroupKFold cross-validation, optimizing mean accuracy, and the selected models are retrained on the full training sets. Table I summarizes the model configurations and decision rules. LSTM receives trace-level feature vectors as single-step inputs rather than raw syscall sequences, while OCSVM and IF are trained exclusively on benign samples. PR-AUC and ROC-AUC are computed from continuous prediction or anomaly scores. The metrics F1-score, Matthews Correlation Coefficient, accuracy, precision, recall and Cohen’s kappa use the fixed decision rules shown in Table II. These thresholds are not changed using validation data or test data. The analysis of zero-day attacks follows an evaluation process.

TABLE II MODEL CONFIGURATIONS AND DECISION RULES

Model	Settings	Training / Selection	Decision Rule
LR	LR = 0.05; 300 iter.; L2 = 10^{-4}	Utility baseline; train-only scaling	$s \geq 0.5$
RF	200 trees; balanced-sub.; seed = 42	Nested 5×3 SGKF; AP; $n_{\text{est}} \in \{100, 200\}$, $\text{depth} \in \{10, \text{None}\}$	$p \geq 0.5$
SVM	RBF; balanced; probability; seed = 42	Nested 5×3 SGKF; AP; $C \in \{1, 10\}$	$p \geq 0.5$
LSTM	32 LSTM; dropout = 0.2; dense = 16; Adam; 15 epochs; batch = 64	Single-step trace input; class-weighted loss	$p \geq 0.5$
OCSVM	RBF; $\nu = 0.1$	Benign-only training	Detector decision
IF	100 trees; seed = 42	Benign-only training	Detector decision

VII. RESULTS AND DISCUSSION

Table II gives an overview of the dataset validation results. There were no records, no missing values and no features that encoded sample identity, class labels or data splits. The dataset includes 22 CVE-based or scenario-based instances across nine attack families and a separate Synthetic Unseen category. It was tested using 44 groups and nested 5×3 group cross-validation. The PR-AUC values were 0.9720 for to-late testing and 0.9487 for late-to-early testing. Cross-environment evaluation reached 0.9900. The reported 95% confidence intervals were 0.8809–0.9972 for PR-AUC, 0.9531–0.9997 for ROC-AUC, 0.8415–0.9764 for F1-score and 0.8323–0.9741, for MCC.

TABLE II

Multi-Stage Validation Results for Dataset Quality

Evaluation Criterion	Result
Invalid records	0
Missing-value rate	0.00%
Metadata/sample-identity features	0
Features encoding split/class information	0
Independent groups	44
Nested group CV configuration	5×3
Attack families (9 known + Synthetic Unseen category)	10
CVEs / attack scenarios	22
Early → Late PR-AUC	0.9720
Late → Early PR-AUC	0.9487
Highest PR-AUC (cross-environment)	0.9900
PR-AUC, 95% CI	0.8809–0.9972
ROC-AUC, 95% CI	0.9531–0.9997
F1, 95% CI	0.8415–0.9764
MCC, 95% CI	0.8323–0.9741

Table III shows that models that were trained with labels performed better than models that were not trained with labels. Random Forest had the PR-AUC value of 0.9558 and the best ROC-AUC value of 0.9845. Support Vector Machine

had the F1-score of 0.9139 and the best MCC value of 0.9065. On the hand One-Class SVM and Isolation Forest had much lower PR-AUC values of 0.3758 and 0.0953. The Friedman test with a p value of 0.002812 shows that there are significant differences, in the performance of the models that were compared.

TABLE III

Comparative Performance of Supervised and Unsupervised Detection models

Model	Learning Paradigm	PR-AUC	ROC-AUC	F1-score	MCC
Random Forest	Supervised	0.9558	0.9845	0.9006	0.8954
SVM	Supervised	0.9277	0.9647	0.9139	0.9065
LSTM	Supervised	0.9197	0.9631	0.8480	0.8325
One-Class SVM	Unsupervised	0.3758	0.7193	0.3343	0.2610
Isolation Forest	Unsupervised	0.0953	0.4998	0.1408	0.0239

Figure 2 compares the detection performance after removing each feature group. Removing the time-related features made no difference, to the PR-AUC score. It went up a little from 0.9547 to 0.9552. When both time-related and structure-related features were taken away the PR-AUC dropped to 0.8926. The biggest drop happened when time-related structure-related and behavior-related features were all removed. The PR-AUC then went down to 0.6729. The behavioral features are thus the most important features for distinguishing attack from benign activity in this dataset. Structural features also contribute to the detection performance, whereas the temporal features add relatively little when used with the other two groups.

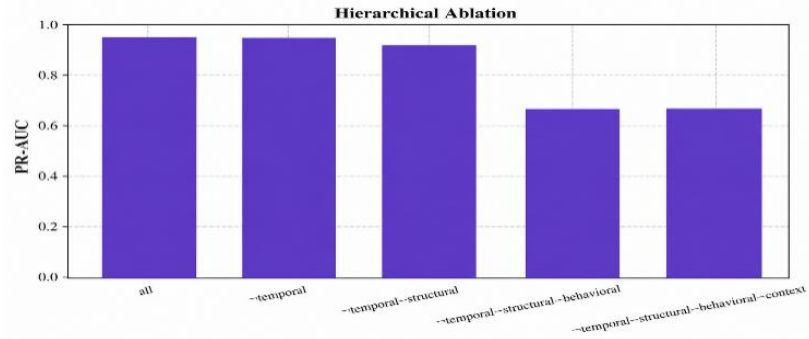


Fig. 2. Feature Ablation study.

Figure 3 gives the scores obtained for the different dataset quality criteria. Integrity and leakage safety received high scores, as did several of the generalization measures. The scores for feature quality and separability were lower. The difference between these scores is consistent with the fact that each criterion evaluates a different aspect of the dataset. No single score captures all of these properties.

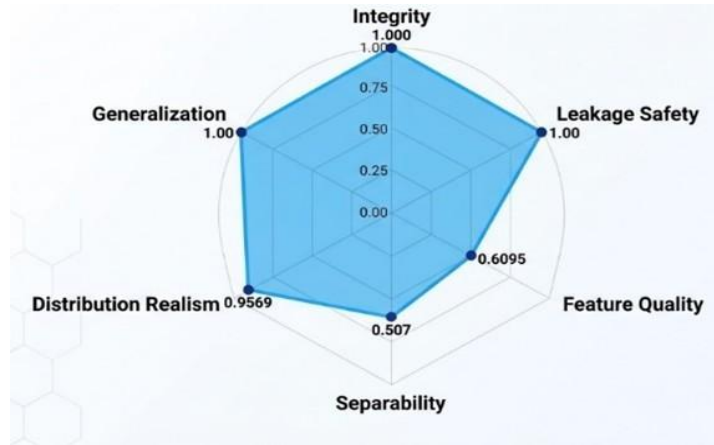


Fig. 3. Multi-dimensional quality assessment of the CHASE dataset

Removing the time-related features made no difference, to the PR-AUC score. It went up a little from 0.9547 to 0.9552. When both time-related and structure-related features were taken away the PR-AUC dropped to 0.8926. The biggest

drop happened when time-related structure-related and behavior-related features were all removed. The PR-AUC then went down to 0.6729.

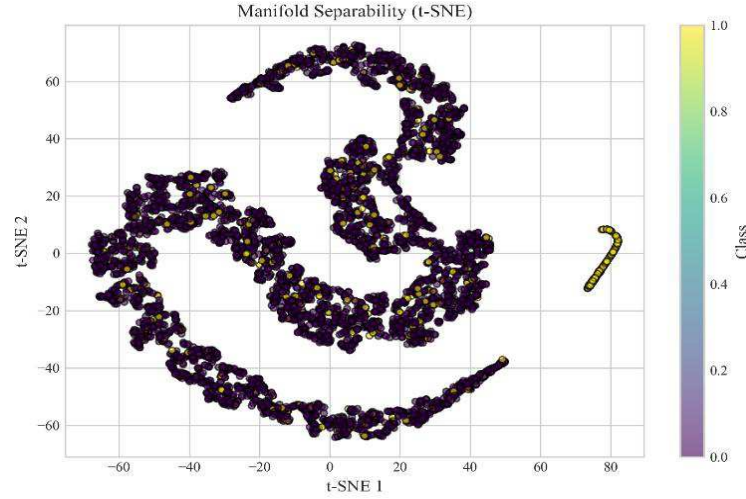


Fig. 4. Analysis of structural separability and spatial distribution of data classes using the t-SNE algorithm

Table IV, gives a summary of how the system detects attacks from several key attack families in CHASE. In the case of Container Escape (CVE-2024-21626) the model was able to spot every attack. It achieved a detection rate of 1.0000 and an F1-score of 0.9362. These figures demonstrate that the model consistently detects this attack scenario. Table V shows how the system generalizes using a Leave-One-Family-Out (LOFO) test. In that test one attack family is omitted from the training data. Then used as a separate test set. The model kept high PR-AUC values for most families that were not part of the training set. The scores ranged from 0.8204 for the Web Injection family to 1.0000, for other families. These outcomes suggest that the behavioral representation captures characteristics that apply to many attack families, not just one.

TABLE IV

ATTACK-LEVEL DETECTION PERFORMANCE ACROSS REPRESENTATIVE CONTAINER-ORIENTED ATTACKS

Attack Family	Attack / Scenario	CVE / Scenario ID	Detection Rate	F1-score	PR-AUC
Container Escape	Container Escape	CVE-2024-21626	1.0000	0.9362	0.9966
Container Escape	Docker Authentication Bypass	CVE-2024-41110	1.0000	0.8800	1.0000
Command Injection	OS Command Injection	CVE-2019-15107	0.9318	0.9011	0.9694
Deserialization	Deserialization RCE	CVE-2020-9483	1.0000	0.8800	0.9632
Resource Abuse	Resource Exhaustion	Resource-Exhaustion	1.0000	0.8750	0.9642
Resource Abuse	Endpoint DoS	GPU-DoS	0.9524	0.8511	0.9440

TABLE V

LOFO PERFORMANCE ACROSS KNOWN ATTACK FAMILIES

Held-Out Attack Family	PR-AUC
Container Escape	0.9950
Web Injection	0.8204
Path Traversal	0.9563
Network Interception	1.0000
Resource Abuse	1.0000
Persistence	0.9823
Command Injection	1.0000
Deserialization	1.0000
Exfiltration	1.0000

Overall the results show that CHASE offers a benchmark for intrusion detection based on containers. The dataset showed quality across verification analyses and helped in evaluating both supervised and unsupervised models in different generalization settings. The ablation results showed how different feature groups contributed, while the lower performance of models shows the difficulties of finding container attacks without labeled data. Table VI shows the differences, between CHASE and previous HIDS datasets including container context, information related to containers and modern attack scenarios. CHASE also allows for controlled evaluation using Synthetic traces; however this setting looks at behavioral novelty and should not be seen as real-world zero-day detection.

TABLE VI
QUALITATIVE COMPARISON OF DATASET CHARACTERISTICS BETWEEN THE PROPOSED DATASET AND REPRESENTATIVE BENCHMARK DATASETS

Feature	ADFA-LD	NGIDS-DS	LID-DS 2021	CHASE (Proposed)
Release Year	2013	2019	2021	2026
Primary Data Source	System Calls	System Calls	System Calls + Context	Behavioral System Calls + Rich Context
Operating System	Linux	Linux	Linux	Linux (Containerized)
Container-aware Environment	×	×	×	✓
Docker Support	×	×	×	✓
Kubernetes Support	×	×	×	✓
Cloud-native Environment	×	Limited	×	✓
System Call Sequences	✓	✓	✓	✓
System Call Arguments	×	Limited	✓	✓
Return Values	×	Limited	✓	✓
Process Metadata	Limited	Limited	✓	✓
Thread-level Information	×	×	✓	✓
Timestamp Information	Limited	✓	✓	✓
Behavioral Context	Limited	Intermediate	Strong	Comprehensive
Real CVE-based Attacks	Limited	Limited	✓	✓ (Recent CVEs)
Modern Attack Coverage	×	Limited	Limited	✓
Container Escape Attacks	×	×	×	✓
Container Runtime Attacks	×	×	×	✓
Cloud-native Attack Scenarios	×	×	×	✓
.	×	Limited	Intermediate	✓
Attack Diversity	Limited	Intermediate	Strong	Broad
Behavioral Realism	Intermediate	Intermediate	Strong	Behaviorally Rich
Suitable for Deep Learning	Intermediate	Strong	Strong	Strong
Suitable for Container Security Research	×	×	×	✓

VIII. CONCLUSION

CHASE is a dataset that helps research, evaluation and comparison of host-based intrusion detection methods in containerized settings. By giving different behavioral traces and rich context of how the system runs CHASE can be used to test current detection methods and to build new approaches, for container safety. CHASE joins system-call sequences with time data, process identifiers, arguments, return values and container-level context in one view. CHASE also records attack scenarios that show the real complexity of container workloads.

REFERENCES

- [1]. N. Joraviya, B. N. Gohil, and U. P. Rao, "DL-HIDS: Deep learning-based host intrusion detection system using system calls-to-image for containerized cloud environment," *The Journal of Supercomputing*, vol. 80, no. 9, pp. 12218–12246, 2024, doi: 10.1007/s11227-024-05895-3.
- [2]. N. Joraviya, B. N. Gohil, and U. P. Rao, "Ab-HIDS: An anomaly-based host intrusion detection system using frequency of N-gram system call features and ensemble learning for containerized environment," *Concurrency and Computation: Practice and Experience*, vol. 36, no. 23, Art. no. e8249, 2024, doi: 10.1002/cpe.8249.

- [3]. T. Vyšniūnas, D. Čeponis, N. Goranin, and A. Čenys, "Risk-based system-call sequence grouping method for malware intrusion detection," *Electronics*, vol. 13, no. 1, Art. no. 206, 2024, doi: 10.3390/electronics13010206.
- [4]. Z. Chen, M. Simsek, B. Kantarci, M. Bagheri, and P. Djukic, "Machine learning-enabled hybrid intrusion detection system with host data transformation and an advanced two-stage classifier," *Computer Networks*, vol. 250, Art. no. 110576, 2024, doi: 10.1016/j.comnet.2024.110576.
- [5]. H. Satilmis, S. Akleylek, and Z. Y. Tok, "A systematic literature review on host-based intrusion detection systems," *IEEE Access*, vol. 12, pp. 27237–27266, 2024, doi: 10.1109/ACCESS.2024.3367004.
- [6]. L. Arnoud, V. Breux, P.-H. Thevenon, and É. Gaussier, "SoK: An in-depth analysis of intrusion detection systems based on system calls," *Journal of Cybersecurity and Privacy*, vol. 6, no. 3, Art. no. 99, 2026, doi: 10.3390/jcp6030099.
- [7]. R. K. Cunningham, R. P. Lippmann, D. J. Fried, S. L. Garfinkel, I. Graf, K. R. Kendall, S. L. Webster, D. Wyschogrod, and M. A. Zissman, "Evaluating intrusion detection systems without attacking your friends: The 1998 DARPA intrusion detection evaluation," in *Proc. 3rd Conf. and Workshop on Intrusion Detection and Response*, 1999.
- [8]. G. Creech and J. Hu, "Generation of a new IDS test dataset: Time to retire the KDD collection," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2013, pp. 4487–4492, doi: 10.1109/WCNC.2013.6555301.
- [9]. J. Hu, "ADFA IDS datasets," UNSW Sydney, 2013. [Online]. Available: <https://research.unsw.edu.au/projects/adfa-ids-datasets>. Accessed: Feb. 4, 2026.
- [10]. G. Creech and J. Hu, "A semantic approach to host-based intrusion detection systems using contiguous and discontinuous system call patterns," *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 807–819, Apr. 2014, doi: 10.1109/TC.2013.13.
- [11]. M. Pendleton and S. Xu, "A dataset generator for next generation system call host intrusion detection systems," in *Proc. IEEE Military Commun. Conf. (MILCOM)*, 2017, pp. 231–236, doi: 10.1109/MILCOM.2017.8170835.
- [12]. J. H. Ring IV, C. M. Van Oort, S. Durst, V. White, J. P. Near, and C. Skalka, "Methods for host-based intrusion detection with deep learning," *Digital Threats: Research and Practice*, vol. 2, no. 4, Art. no. 26, pp. 26:1–26:29, 2021, doi: 10.1145/3461462.
- [13]. M. Grimmer, T. Kaelble, and E. Rahm, "Improving host-based intrusion detection using thread information," in *Emerging Information Security and Applications, Communications in Computer and Information Science*, vol. 1403, W. Meng and S. K. Katsikas, Eds. Cham, Switzerland: Springer, 2022, pp. 159–177, doi: 10.1007/978-3-030-93956-4_10.
- [14]. M. Grimmer, T. Kaelble, F. Nirsberger, E. Schulze, T. Rucks, J. Hoffmann, and E. Rahm, "Dataset report: LID-DS 2021," in *Critical Information Infrastructures Security: CRITIS 2022, Lecture Notes in Computer Science*, vol. 13723. Cham, Switzerland: Springer, 2023, pp. 63–73, doi: 10.1007/978-3-031-35190-7_6.